

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МОЭВМ**

**отчет**  
**по лабораторной работе №2**  
**по дисциплине «Алгоритмы и структуры данных»**  
**Тема: Рекурсивная обработка иерархических списков**

Студент гр.8304

\_\_\_\_\_

Холковский К.В.

Преподаватель

\_\_\_\_\_

Фирсов М.А.

Санкт-Петербург

2019

## **Задание.**

### **18-й Вариант**

Логическое, вычисление, добавить 4-ую операцию (которая может принимать 2 аргумента)(был добавлен XOR), префиксная форма

## **Цель работы.**

Решить полученную задачу, используя иерархические списки.

Получения навыков работы с нелинейными структурами данных.

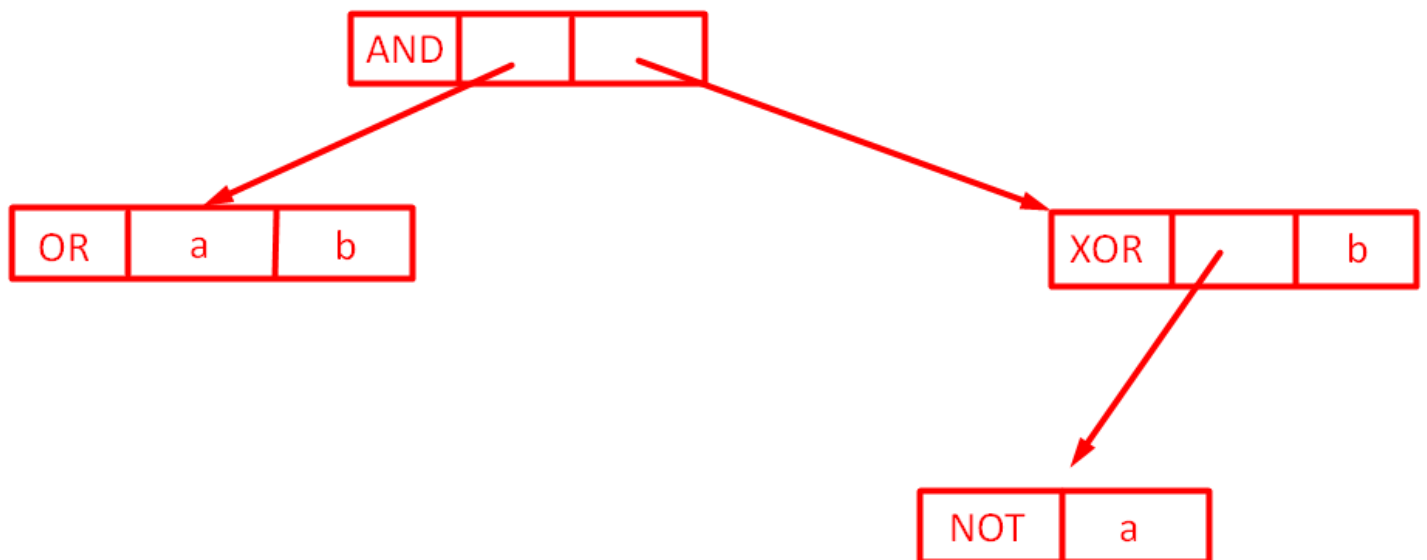
## **Описание алгоритма.**

В начале, программа получает строку с переменными и их значениями формата  $((x_1 \ c_1)(x_2 \ c_2) \dots (x_i \ c_i))$  и записывает их в словарь `std::map<std::string, bool> dict`. Далее идет обработка строки – выражения формата  $(\langle \text{операция} \rangle \langle \text{аргументы} \rangle)$  функцией `void func(Node*&, std::string const&, std::map<std::string, bool> const&)`, которая внутри каждого дочернего выражения создает узлы и записывает их в родительский узел. Также внутри этой функции используется функция для создания узла или атома `void mkAtom(std::string const&, int&, Node*&, std::map<std::string, bool> const&)`. После отработки функции, имеем готовый иерархический список.

## **Описание функций и структур данных.**

- `Struct Node;` - хранит данные о дочерних узлах и операцию, либо значение атома.
- `bool Node::evaluate()` - если `std::variant<std::string, bool> Node::val` хранит атом, то возвращает значение атома, иначе возвращает значение полученное при применении оператора к узлам, хранящимся в `std::variant<std::pair<Node*, Node*>, Node*> argv`
- `void func(Node*&, std::string const&, std::map<std::string, bool> const&);`  
Создает узел из двух дочерних узлов.
- `void mkAtom(std::string const&, int& ind, Node*&, std::map<std::string, bool> const&)` либо создает атом, либо говорит из какой строки функции `func` следует сделать узел.

Для выражения  $(\text{AND}(\text{OR } a \ b)(\text{XOR}(\text{NOT } a) \ b))$  иерархический список будет выглядеть так:



### Выводы.

Для решения полученной задачи нецелесообразно было использовать иерархический список, по - моему лучше было использовать стек.

### Тестирование

```
For filename:Tests/Test1
Для значений:
Dimon : 0
Kris : 1
Leha : 1
И для выражения: (OR (AND Leha Kris)(AND Dimon Kris))
Ответ: 1

For filename:Tests/Test2
Для значений:
ar : 0
br : 1
И для выражения: (AND ar)
Operator does not match expression

For filename:Tests/Test3
Для значений:
Oleg : 1
Serega : 1
И для выражения: (XOR (NOT Serega)Oleg)
Ответ: 1
```

Пример вывода программы

<b>((Leha 1)(Dimon 0)(Kris 1)) (OR (AND Leha Kris)(AND Dimon Kris))</b>	<b>1</b>
<b>fasfafdsaaf dasdadada</b>	<b>Invalid operator!</b>
<b>afsafafafasf (AND a b)</b>	<b>a - not declared</b>
<b>((Serega 1)(Oleg 2)) (XOR (NOT Serega)Oleg)</b>	<b>1</b>

### Исходный код

```
#include <iostream>
#include <variant>
#include <map>
#include <algorithm>
#include <fstream>
#include <map>

struct Node{
~Node()
{
if(std::holds_alternative<Node*>(argv))
delete std::get<Node*>(argv);
else
{
delete std::get<std::pair<Node*, Node*>>(argv).first;
delete std::get<std::pair<Node*, Node*>>(argv).second;
}
}

bool evaluate()
{
if (std::holds_alternative<std::string>(val))
{
if (std::get<std::string>(val) == "AND")
return std::get<std::pair<Node*, Node*>>(argv).first->evaluate() &&
std::get<std::pair<Node*, Node*>>(argv).second->evaluate();
if (std::get<std::string>(val) == "OR")
return std::get<std::pair<Node*, Node*>>(argv).first->evaluate() ||
std::get<std::pair<Node*, Node*>>(argv).second->evaluate();
if (std::get<std::string>(val) == "XOR")
return (std::get<std::pair<Node*, Node*>>(argv).first->evaluate() ||
std::get<std::pair<Node*, Node*>>(argv).second->evaluate())
```

```

        && !(std::get<std::pair<Node*, Node*>>(argv).first->evaluate() &&
std::get<std::pair<Node*, Node*>>(argv).second->evaluate());
        if (std::get<std::string>(val) == "NOT")
            return ! std::get<Node*>(argv)->evaluate();
    }
else
    return std::get<int>(val);
}

std::variant<std::pair<Node*, Node*>, Node*> argv;
std::variant<int, std::string> val; //std::variant<bool, std::string> val;
};

typedef struct Node Node;

void func(Node*&, std::string const&, std::map<std::string, bool> const&);

void mkAtom(std::string const& str, int& ind, Node*& arg, std::map<std::string, bool>
const& dict){
    if (str[ind] == '(')
    {
        int tmp = ind, error = 0;
        std::string tmp_s = "";
        while (1) //нахождение соот-ей ')'
        {
            tmp_s += str[tmp];
            tmp++;
            if (str[tmp] == '(')
                error++;
            if (str[tmp] == ')')
                error--;
            if (error < 0)
                break;
        }
        tmp_s += str[tmp];
        func(arg, tmp_s, dict);
        ind = tmp + 1;
    }
else
    {
        std::string tmp = "";
        while (str[ind] != ' ' && str[ind] != '(' && str[ind] != ')')
            tmp += str[ind++];
        auto tr=dict.find(tmp);
    }
}

```

```

if(tr==dict.end())
{
    std::cout<<tmp<<" - not declared";
    exit(1);
}

arg->val = tr->second;
}
}

void func(Node* & point, std::string const& str, std::map<std::string,bool> const& dict)
{
    int ind = 0;
    Node* arg1 = new Node;
    Node* arg2 = new Node;
    while (str[ind] == '(' || str[ind] == ' ')//перейти к оператору
        ind++;
    std::string oper="";
    while(str[ind] != ' ' && str[ind] != '(') //}считывание оператора
        oper+=str[ind++];
    if (oper != "AND" && oper != "OR" && oper != "NOT" && oper != "XOR")//проверка оператора
    {
        std::cout << "Invalid operator!\n";
        delete arg1;
        delete arg2;
        exit(1);
    }
    point->val = oper;
    while(str[ind] == ' '){ind++;} //переход к первому аргументу или скобочке
    mkAtom(str, ind, arg1, dict);
    while (str[ind] == ' '){ind++;} //переход к следующему аргументу или скобочке
    if (str[ind] == ')')
        //проверка унарности
    {
        if (std::get<std::string>(point->val) == "NOT")
        {
            point->argv = arg1;
            delete arg2;
            return;
        }
    }
    else
    {
        std::cout << "Operator does not match expression\n";
        exit(1);
    }
}

```

```

}
}
mkAtom(str, ind, arg2, dict);
point->argv = std::make_pair(arg1, arg2); //сохранение дочерних узлов в родительском
}

int main(int argc, char* argv[]){
if(argc==1)
{
std::map<std::string,bool> dict;
char* key=new char[20];
int val;
std::cout<<"Введите список значений переменных"<<std::endl;
scanf("(");
while(scanf("(%s %d)",key,&val)==2)
    dict.insert({key,val});
delete[] key;
std::string s;
std::getline(std::cin,s);
std::cout<<"Введите выражение"<<std::endl;
std::getline(std::cin,s);
Node* point = new Node;
func(point, s, dict);
std::cout << point->evaluate() << std::endl;
delete point;
}
else
{
std::cout<<"For filename:"<<argv[1]<<std::endl;
std::ifstream in(argv[1]);
if (!in.is_open()){
    std::cout<<"Can't open file"<<std::endl;
    return 0;
}
if (in.eof()){
    std::cout<<argv[1]<<"is empty File"<<std::endl;
    return 0;
}
FILE* file = fopen(argv[1], "r");
std::map<std::string,bool> dict;
char* key=new char[20];
int val;
fscanf(file, "(");

```

```
while(fscanf(file,"%s %d",key,&val)==2)
    dict.insert({key,val});
delete[] key;
fclose(file);
std::string s;
std::getline(in,s);
std::getline(in,s);
Node* point = new Node;
std::cout<<"Для значений: "<<std::endl;
for(auto tr = dict.begin();tr != dict.end(); ++tr)
    std::cout<<tr->first<<" : "<<tr->second<<std::endl;
std::cout<<"И для выражения: "<<s<<std::endl;
func(point, s, dict);
std::cout <<"Ответ: "<< point->evaluate() << std::endl;
delete point;
}
return 0;
}
```